L'architecte et le chaos

Jacques PRINTZ, jacques.printz@cnam.fr, Professeur Emérite
Conservatoire National des Arts et Métiers – Chaire de génie logiciel

Sommaire

COMMUNIQUER A L'AIDE DE METAPHORES 1
ORGANISER NOS CONNAISSANCES 5
LE PATRIMOINE APPLICATIF DANS LE LANGAGE DES INFORMATICIENS 8
LE PATRIMOINE APPLICATIF DANS LE LANGAGE DU BIBLIOTHECAIRE 11
LA PSYCHOLOGIE DU PROGRAMMEUR ET L'IMPACT DE LA DEMATERIALISATION DES SUPPORTS PHYSIQUES DE LA PROGRAMMATION 13
L'INFORMATION DANS LE NANO MONDE, AU DELA DU BIOLOGIQUE 16
INTEGRITE DE LA BIBLIOTHEQUE : PAS SI SIMPLE ! 17
L'ARCHITECTE ET LE CHAOS : ARCHITECTURE DE LA COMPLEXITE 18
L'INGENIEUR ET LE BRICOLEUR 19
BIBLIOGRAPHIE 20

Figures

Le quartier de la Défense *2*
Communication humaine et codage *5*
Bibliothèques *6*
Indexation des connaissances et navigation *7*
Reconstruction *10*
Propagation des erreurs et métastases *12*
Temps moyen de correction des défauts *14*
Courbes de maturité *15*
Le vrai patrimoine applicatif *16*

Avertissement : nous utilisons le terme systèmes d'information (SI) de façon extensive, en y intégrant les SI classiques du monde des services et du tertiaire, mais également les systèmes industriels technologiques, autrefois séparés, mais qui désormais interopèrent quasiment sans limite avec les premiers. Rappelons que le mot informatique fut forgé, il y a plus de 40 ans par la fusion des mots information + automatique, donc est informatique tout ce qui automatise le traitement de l'information digitalisée, quelle que soit la source et la nature de cette information.

Communiquer à l'aide de métaphores

C'est un lieu commun de dire qu'il n'est pas facile de communiquer à propos de l'architecture des systèmes d'information avec tous ceux qui ont affaire à l'informatique mais qui ne sont pas des informaticiens pour autant. C'est même vrai avec des professionnels tant le mot lui-même est connoté différemment selon les disciplines qui l'utilisent : l'architecture d'un réseau n'a qu'un lointain rapport avec une architecture logicielle.
Remarquons d'entrée de jeu pour éviter les banalités que, dès que l'on veut communiquer d'une façon rigoureuse, il faut un langage différencié et précis, adapté à chaque situation. Chaque métier à le sien. Il paraît que les esquimaux ont des dizaines de mots pour parler de la neige, comme nous même nous en avons des dizaines pour parler du vin.

Les communautés des méthodes agiles[1] et de l'eXtreme Programming préconisent l'usage de métaphores pour mieux se comprendre, y compris au sein même de l'équipe, avec les parties prenantes : « *Metaphor. The team develops a common vision of how the program works* » que l'on peut traduire en « L'équipe développe une vision commune du fonctionnement du programme à l'aide de métaphores ». De fait, l'équipe adopte une idéographie[2], avec des signes qui fondent son identité en tant que groupe.
Remarquons qu'à l'aube de l'informatique, les programmeurs d'alors étaient qualifiés de « codeur », terme aujourd'hui péjoratif, mais très vite le terme de programmeur, spécialiste du « langage » informatique, jugé abscons, s'est définitivement imposé, et ce n'est sûrement pas un hasard tant la proximité avec le langage humain est forte.
Il est intéressant de noter que dans d'autres technologies, comme les circuits intégrés[3] ou l'électronique, on n'a jamais éprouvé ce type de besoin. Sans doute faut-il en chercher la cause à la fois dans la nature abstraite et immatérielle du logiciel, mais en même temps dans la relation que le logiciel entretient avec nos actes les plus élémentaires. D'une certaine façon, pour bien comprendre il nous faut matérialiser, avec des objets du monde physique, le contenu de la « boîte noire » logicielle. Pour changer les habitudes, ou opérer des transformations dans la société, les médias utilisent volontiers l'expression « changer de logiciel », sans se douter de la complexité de l'opération et de l'inadéquation foncière de la métaphore.
Toujours est-il que pour communiquer avec son environnement managérial, avec ses clients, l'architecte de système d'information va utiliser des métaphores dont l'une des plus connues est celle de l'architecture au sens littéral des constructions civiles et de l'organisation des villes : c'est la métaphore de l'urbanisation.

Le quartier de la Défense[4]

[1] Voir le site http://agilemanifesto.org/
[2] Pour être parfaitement rigoureux, et fidèle à U.Ecco, le terme exact serait « sémiologie ».
[3] Les concepteurs de circuits, chez les constructeurs, étaient appelés : « logiciens ».
[4] On prête à De Gaulle, survolant le site futur de La Défense, s'adressant à P.Delouvrier, le mot « Mettez moi de l'ordre dans ce bordel » !

L'un des premiers à l'avoir utilisée est John Zachman, dans son article de l'IBM Systems Journal, en 1987, mais toutefois en insistant plus sur la notion de plan, évidement très prégnante et ancienne, dans le monde du BTP.
Il peut paraître paradoxal de comparer une matière « molle » comme le logiciel (en anglais « soft » signifie mou, doux) avec le béton, même si F.Bouygues en parlait comme de la « matière grise ».
Ce qui caractérise le mieux le logiciel est son caractère évolutif, en perpétuelle adaptation, son ubiquité, du moins pour les logiciels intégrés dans les organisations. C'est un peu moins vrai pour les logiciels intégrés dans les équipements, encore que, avec les téléphones portables et autres GPS qui désormais nous connectent tous ... !
C'est même la distinction fondamentale introduite par Von Neumann, dans les architectures qui portent son nom, où le programme enregistré en mémoire est tout aussi facilement modifiable que les données qu'il manipule dans la même mémoire.
En bref, tout le contraire du béton !

Dans les constructions civiles, et plus généralement dans l'ingénierie du dur (la mécanique, l'électrotechnique, l'électronique, l'électromagnétisme, l'aéronautique, etc.) on distingue radicalement le plan, le « blue print », en souvenir d'une technique de reproduction aujourd'hui disparue, avec d'ailleurs les tables à dessins, qui ont structuré les formations de générations d'ingénieurs à grand renfort de géométrie[5], ... et la réalisation proprement dite. Dans ces domaines de l'engineering, il est hors de question de modifier le plan lorsque la réalisation et l'industrialisation sont lancées. En contrepartie, dans le BTP, le réalisateur donne une garantie décennale. Il est responsable et coupable, en cas de défaillance avérée de son fait : c'est un signe de maturité.
Remarque : Dans les sciences de l'ingénieur, la résistance des matériaux occupe une place centrale qui n'a pas, pour le moment, d'équivalent dans les sciences de l'information.

Là encore, tout le contraire de ce que l'on fait dans le traitement de l'information, où idéalement on souhaiterait modifier la programmation[6] jusqu'à la dernière minute.
Dans cette discipline nouvelle encore bien jeune, on confond sans arrêt l'autorité et l'expertise, l'usager de l'informatique et l'informaticien expert. Nous sommes tous des usagers de l'automobile et des moteurs, mais cela ne nous donne aucune compétence en mécanique. Cette confusion des qualifications entre l'usage et le faire, est la source de bien des déconvenues en informatique.

Les protagonistes des méthodes agiles ont fait de la communication par métaphore un véritable étendard ; au sens littéral, « *standard* » en anglais est l'étendard, le blason, que portaient les chevaliers du moyen age en signe de reconnaissance dans les batailles. Mais entre le dire, et le faire, il y a souvent une marge. Trouver de bonnes métaphores est un art difficile, car de fait c'est inventer un langage si l'on veut communiquer effectivement, avec des actes à l'appui.
Le Corbusier[7], qui s'y connaissait en matière d'urbanisation, définissait la ville comme une machine à produire, comme un organisme où l'ordre doit régner. Il a fait l'apologie du plan, et de la géométrie qui en est inséparable, et il admirait les ingénieurs. On a oublié que la géométrie descriptive, inventée par Gaspard Monge, professeur à

[5] Parfois très savante comme les surfaces réglées pour la taille des engrenages, ou les surfaces caustiques en optique pour la taille des lentilles.
[6] Dans l'histoire de l'informatique, la machine de K.Zuse s'appelait « Plan Calcul » (en allemand).
[7] Voir ses livres : *Urbanisme, Vers une architecture, La charte d'Athènes* ou encore *Le modulor*.

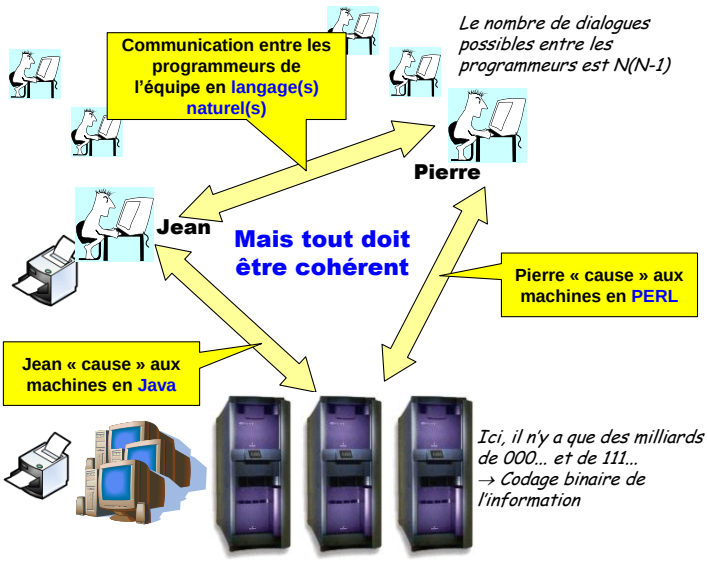
Polytechnique au moment de sa fondation, était considérée comme un secret d'état, et que « l'art du trait » des compagnons qui ont construit nos cathédrales ne s'enseignait que de maîtres à disciples, où le compagnon devait être moralement « qualifié ».

Dans la discipline informatique, ce qui correspond au plan, c'est le modèle, où plutôt les modèles comme ceux proposés par la méthode MERISE dans les années 1970, ou aujourd'hui dans le langage UML. De même, dans le BTP ou la mécanique, il y a de nombreux types de plans, ceux pour communiquer avec les clients, comme les plans en perspectives, et ceux pour travailler, à base de géométrie descriptive. Et même des maquettes que l'on peut encore admirer au musée des Plans et Reliefs, aux Invalides.
Pour circuler dans une ville, il y a une hiérarchie de voies de communications : ruelle, rue, avenue, voie rapide, autoroute, etc. que l'on pourrait retrouver dans la façon dont nous organisons la circulation de l'information dans nos systèmes, via les mémoires partagées et/ou les réseaux, conformément aux performances requises.
En ce sens, la métaphore urbanistique est pertinente.

Le Club Urba EA[8], dans *Urbanisme des SI et Gouvernance*, définit l'urbanisation du SI comme suit : « Urbaniser le système d'information, c'est le simplifier. C'est trouver – avec une volonté d'anticipation – un découpage et des grands principes de construction qui permettront de faire évoluer le système d'information et l'informatique en cohérence avec la stratégie et l'organisation. La métaphore de la ville est le meilleur moyen de comprendre la démarche d'urbanisation. »
Le problème, c'est : « comprendre », mais jusqu'où ? A quelle profondeur ? Et quel éclairage à la fois pertinent dans sa représentation et efficace dans la communication nous apporte la métaphore de la ville ? D'un point de vue concret, au-delà de l'incantation, comment réellement simplifier ? Avec quels critères de mesure ?

Pour ce qui concerne le logiciel proprement dit, sous ses différentes formes : données factuelles (fichiers, bases de données, textes, etc.), données événementielles (événements, messages, etc.), procédures de traitements des informations (les programmes, écrits dans toutes sortes de langages), la métaphore urbanistique atteint rapidement ses limites. Des compléments métaphoriques sont nécessaires si l'on veut continuer à communiquer par métaphores interposées.
Une distinction aussi fondamentale que la syntaxe, i.e. la structure de l'information, et la sémantique, i.e. le sens de cette information, ce qui va permettre d'agir, n'ont aucune correspondance dans le monde des villes. Le changement de forme (ou de code), à sémantique constante, qui est le B.A.–BA des ordinateurs n'a pas non plus d'équivalent. Mais nous comprenons tous cette différence, par l'expérience que nous avons de nos propres langues et de la grammaire. Avec le chinois, qui est une écriture idéographique ce serait différent !
Von Neumann, dès le début avait distingué radicalement le langage interne de la machine, le binaire, du langage externe fait pour communiquer avec les humains, autre distinction fondamentale qui n'a pas d'équivalent dans l'urbanisation. Les tous premiers langages comme FORTRAN et COBOL en étaient de bonnes illustrations. On retrouve également cette distinction dans ce que MERISE appelait modèle conceptuel, pour les métiers, et modèle logique/physique pour les informaticiens.

[8] Voir le site : http://www.urba-ea.org/


Communication entre les programmeurs de l'équipe en langage(s) naturel(s)
Le nombre de dialogues possibles entre les programmeurs est N(N-1)
Pierre
Jean
Mais tout doit être cohérent
Pierre « cause » aux machines en PERL
Jean « cause » aux machines en Java
Ici, il n'y a que des milliards de 000... et de 111...
→ Codage binaire de l'information

Communication humaine et codage

La notion de module (en anglais *building block*), au sens de D.Parnas, redécouvert 25 ans plus tard par J.Sassoon, qui est une notion centrale dès que l'on souhaite une découpe intelligente de l'information qui sous-tend toute la démarche SOA, n'a pas non plus d'équivalent dans la ville, même en allant chercher Le Modulor, de Le Corbusier. Le module informatique est la « brique » de tous nos systèmes d'information. Pour que la construction soit solide, il faut que la brique soit standardisée et de bonne qualité, c'est-à-dire satisfasse aux contraintes externes[9] issues des métiers et/ou des équipements comme la fiabilité, la performance, la facilité d'emploi, la maintenabilité, la sécurité, etc. Quant au côté exécutable de l'information procédurale .... n'en parlons même pas ! Le POS informatique n'est qu'une structure statique qui ne reflète en rien le caractère dynamique et interactif des traitements d'aujourd'hui.

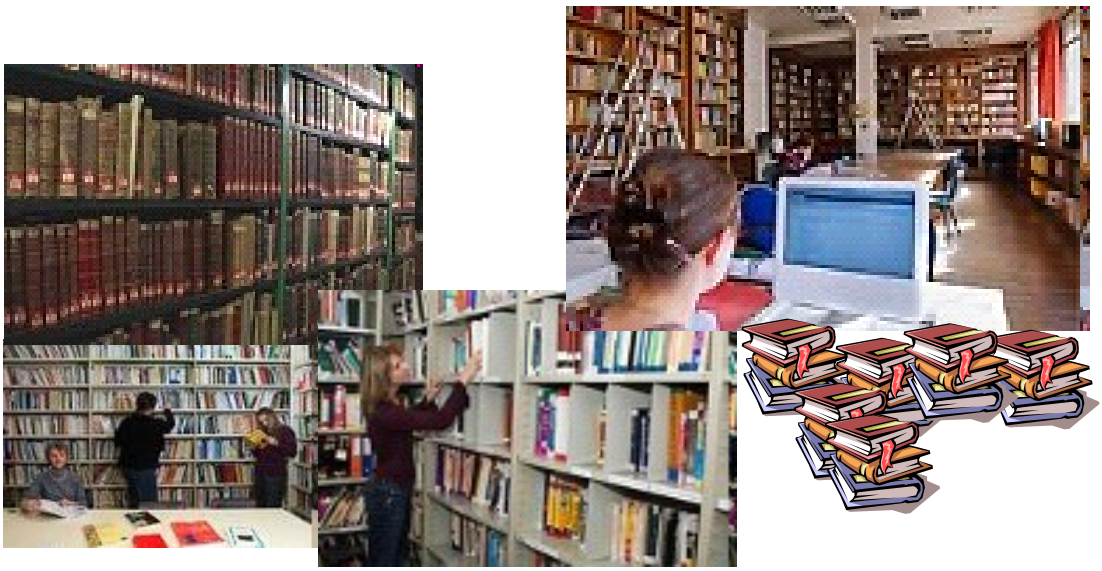
Nous proposons ici de compléter la métaphore de la ville par celle de la bibliothèque, et des livres, qui nous est tout aussi familière. En fait, notre langage, tant écrit que parlé, n'est qu'une vaste métaphore, et de fait un outil irremplaçable pour comprendre ce qui se passe dans la « boite noire ». Notre langage et les textes qui le matérialisent servent à échanger et communiquer des informations, mais aussi à donner et recevoir des ordres, ce que les linguistes appellent des « performatifs », mot dans lequel on retrouve, avec le même sens, notre bon vieux `perform` du COBOL qui sert à « exécuter » une procédure de gestion codée dans le langage COBOL.
Nous avons une relation séculaire avec le livre, objet familier entre tous, que l'on peut mettre dans sa poche, jusqu'aux typographies qui doivent être belles pour faire de la lecture un acte agréable. Nous savons tous qu'écrire un livre n'est pas si facile, qu'il y a une limite en taille, aux alentours de 3 à 400 pages, à la fois pour l'écrivain qui doit rester maître de son sujet, pour l'éditeur qui n'aime pas les gros pavés invendables, et surtout pour le lecteur qui souhaite comprendre et dont il faut ménager la mémoire.

Organiser nos connaissances

Tout d'abord un constat avec lequel nous sommes tous d'accord : le système d'information d'une entreprise n'est rien d'autre que la bibliothèque des ses

[9] La norme ISO/CEI 9126, *Caractéristiques qualité des produits logiciels*, donne une nomenclature détaillée de ces contraintes.

connaissances factuelles, procédurales et événementielles, de ses savoir-faire, le reflet de sa stratégie et de son histoire, ...
Précisons d'abord le sens du mot information dont l'emploi remonte à Aristote, en passant par les philosophes scolastiques de notre moyen age, comme Saint Thomas d'Acquin, grand professeur à Paris dans ce qui n'était pas encore la Sorbonne, ou le Frère Guillaume d'Ockham qui a inspiré U.Ecco dans son merveilleux livre *Au nom de la rose* dont le personnage centrale est la bibliothèque du monastère et son *scriptorium*.
L'information est ce qui donne forme, ce qui anime la matière jugée inerte par les philosophes grecs, assertion contestable en physique quantique, mais c'est 2.500 ans plus tard. Par analogie, sans forcer le trait, on peut dire que le système d'information est ce qui va donner vie à l'information détenue par la masse des acteurs, individus et/ou organisations, qui sont les unités actives de l'entreprise. Sans information, l'entreprise n'est qu'une masse humaine informe incapable de communiquer et d'agir efficacement. Dans ce sens, il serait intéressant d'explorer la métaphore de l'armée et de l'action militaire. Les plans de nombreuses villes européennes gardent encore dans leurs « gènes » la structure d'un camp de légionnaires romains.
Voyons comment cette bibliothèque est organisée !



© Photothèque du Cnam

Bibliothèques

Quand on entre dans une grande bibliothèque, deux choses nous frappent instantanément :

- Le système de classement des livres avec ses nombreux index thématiques qui vont permettre à l'usager de rechercher ce qui l'intéresse. Classer est un attribut distinctif de notre intelligence et des capacités de notre cerveau. C'est un adressage, l'analogue des guides et plans avec le dictionnaire des rues que nous utilisons couramment pour nous retrouver dans nos villes.
- Le système de rangement des livres qui impressionne par son énormité. Une bibliothèque d'un million de livres, c'est environ 30 km de rayonnages, soit en rangeant les livres sur 10 niveaux un pan de mur de 3 km, que l'on peut

structurer en 3.000 cellules de H:2,5×L:1×P:0,3 en mètres[10].

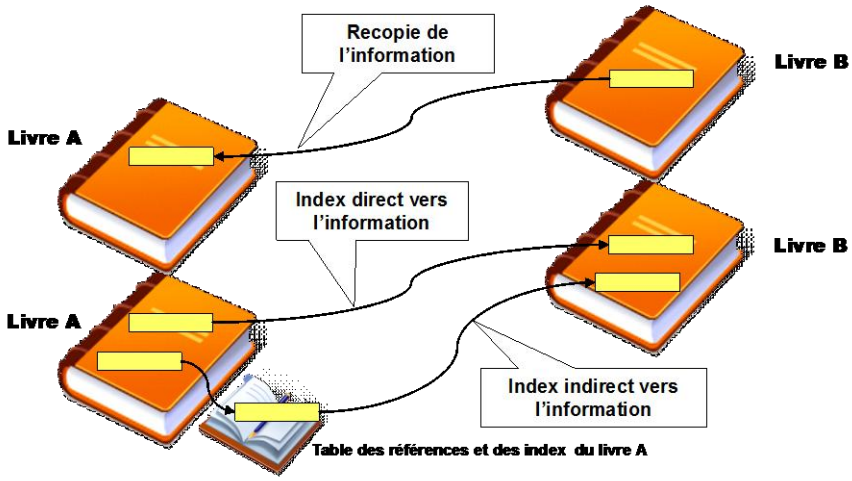
Pour le classement, la contrainte d'emploi est ergonomique car l'usager va manipuler lui-même les index qui doivent donc être à son échelle.

Pour aller chercher les livres il va falloir des opérateurs humains et des machines, en grand nombre qui devront tous maîtriser le plan d'adressage.

Pour les très grandes bibliothèques comme la bibliothèque de France, celle du British muséum, ou celle du Congrès, ce seront des immeubles ...

Pour ce qui concerne les connaissances à organiser, on a immédiatement trois types de problématiques :

- Comment représenter les connaissances ? C'est le problème des écritures et des langues, et de leur interprétation, car ce qui distingue l'information du bruit, c'est l'action. L'information permet d'agir, c'est le performatif dont on a déjà parlé, et si possible agir de façon cohérente.
- Comment classer et organiser les connaissances, regrouper et mutualiser, à commencer par les dictionnaires et les encyclopédies, en respectant les règles de l'ergonomie humaine, dans l'espace et dans le temps ?
- Comment ranger les supports écrits ? En faisant l'hypothèse que tout est papier, ce qui n'était pas le cas à l'époque de Platon et Aristote où il fallait apprendre par cœur !



Indexation des connaissances et navigation

Ces trois problématiques seront immédiatement suivies de deux autres, liées en fait au stockage autre que cérébral et à l'évolution du monde :

- Comment mettre à jour la bibliothèque des connaissances en y insérant de nouvelles connaissances, en évitant les incohérences résultant des citations, forme ancienne des couper/coller, et faire évoluer le réseau sémantique global des connaissances ?
- Comment effacer proprement des connaissances périmées, voire fausses, et récupérer l'espace laissé vacant par les livres mis au pilon ou aux archives ?

La gestion de nos connaissances en bibliothèques préfigure ce que nous faisons tous les jours en informatique : **classer**, **retrouver**, **modifier**, **déplacer**, **effacer**.

[10] On retrouve cette structure en pages sur les disques, avec des tailles de pages de 1, 2 ou 4 kilo-octets qui sont les unités d'alimentation vers la mémoire RAM de la machine.

Les programmes et les données qui constituent la « matière » même de nos SI sont clairement à ranger dans nos connaissances. La seule différence est que l'ordinateur va se substituer au cerveau humain en tant que système de traitement de ces connaissances, mais sans confondre leurs capacités respectives, comme John von Neumann nous l'avait expliqué dans son dernier livre *The computer and the brain*, édité peu de temps après sa mort en 1957. Seul le cerveau humain comprend.

Remarque : A propos des classements nous n'avons généralement pas conscience de la formidable complexité qui se cache derrière cet acte anodin. Pour ranger 14 objets dans seulement 4 boîtes, le nombre de possibilités est supérieure à 10 millions[11] ! Donc, sans règle entre les classificateurs, c'est le chaos immédiat. Cette problématique est l'essence de l'architecture des données.

Le patrimoine applicatif dans le langage des informaticiens

A l'époque où la communauté des informaticiens était composée majoritairement d'ingénieurs et de mathématiciens/physiciens reconvertis, jusqu'au début des années 80, l'habitude a été prise de mesurer la taille des programmes en nombre d'instructions écrites par les programmeurs. Le support matériel de ces écritures était la carte perforée de 80 colonnes, aujourd'hui disparue, correspondant à une ligne de texte. La taille d'un programme était donc quelque chose de physiquement tangible en mètre linéaire et en poids. Un programme de 20.000 instructions, c'était 20.000 cartes perforées, soit 5 bacs de cartes de 1 mètre de long, que le programmeur pouvait porter, pour un poids de 15 kg. Cela se voyait, c'était lourd, et les entreprises d'alors qui avaient du matériel informatique entretenaient des équipes de perforatrices (de formation dactylo) chargées de produire ces cartes à partir des feuilles de programmation écrites par les programmeurs.

Une carte perforée c'est une ligne de texte dans un livre aux normes de l'édition, en relation avec les contraintes ergonomiques de notre champ visuel car il faut que notre œil puisse lire et que notre cerveau comprenne, en un seul coup d'oeil.

Une page normale d'un livre normal, respectant nos contraintes ergonomiques de vision et de manipulation à la main, c'est environ 50 lignes. L'usage des livres fait de la double page une unité physiologique de compréhension pour notre mémoire immédiate. Ce qui fait qu'un livre de 400 pages, c'est environ 20.000 lignes de texte écrit.

Par analogie, un programme de 20.000 instructions est équivalent en taille matérielle à un livre de 400 pages, à propos duquel on peut se poser deux questions :

- Combien de temps pour le concevoir, l'écrire et le corriger ?
- Combien de temps pour le lire et le comprendre ? La encore, en respectant des règles ergonomiques 7 rubriques par niveau hiérarchique et pas plus de 3 niveaux, soit tout de même 7×7×7 possibilités de structuration.

Un bon lecteur qui lit en moyenne une quarantaine de page à l'heure, en comprenant ce qu'il lit, mettra donc 10 heures à lire le livre, sachant qu'on ne peut pas lire 10 heures d'affilées. Si le livre est un peu technique, il faudra diviser la vitesse par un facteur 5-10.

Remarque : mon dernier livre, *Architecture logicielle*, 450 pages, m'a coûté un peu plus de deux ans de travail, quasiment tous les jours mais pas à temps plein.

[11] Les formules de calcul sont des classiques de la théorie des ensembles, théorie qui joue, avec la logique, un rôle central dans l'architecture des données et des systèmes. Soit : ensemble des partitions de N individus à classer = 2^N ; ensemble des rangements possibles de N individus dans k boîtes = $\frac{k^N}{k!}$.

On trouve sans difficulté des statistiques sérieuses[12] sur la taille des programmes exprimées en lignes de code source (LS), avec une méthode de comptage codifiée par le *Software Engineering Institute* de l'université Carnegie Mellon, à Pittsburgh. En voici quelques exemples :

- Le système d'exploitation LINUX : 2,3 millions de LS
- Le système d'exploitation Windows NT : 3,8 millions de LS
- Le système d'exploitation GCOS7 de Bull[13] : 4,034 millions de LS
- Word 4.0 de la suite Microsoft Office : 256.378 LS
- Excel 4.0 de la suite Microsoft Office : 851.468 LS
- Un compilateur qui effectue une traduction du langage FORTRAN en langage machine : environ 250.000 LS
- Le programme de mensualisation/recouvrement de nos impôts : environ 1 million de LS
- Les logiciels de la navette spatiale : 2,2 millions de LS
- Le système de combat du porte-avions Charles de Gaulle : environ 5 millions de LS
- Le logiciel pour la guerre des étoile (initiative de défense stratégique du président Reagan) : environ 10 millions de LS
- Le patrimoine applicatif d'une société comme Amadeus : environ 150 millions de LS

Il est intéressant de ramener ces masses qui ne disent pas grand-chose, y compris pour l'informaticien patenté, à l'échelle humaine qui est celle des livres que nous sommes capables d'écrire, de lire et de comprendre, soit 400 pages. Il suffit de diviser les chiffres précédents par 20.000, et cela donne :

- LINUX : 115 livres
- Windows NT : 190 livres
- GCOS7 : 202 livres
- La guerre des étoiles : 500 livres
- Amadeus : 37.500 livres

Encore ne s'agit-il que du programme « brut », auquel il faudrait rajouter la documentation qui explique le pourquoi et le comment du programme, et tous les tests qui ont servi à le valider, soit un triplement de la taille. Sans ces textes le programme est incompréhensible, sauf à le réécrire, ce qui n'est pas le but.

Ramené à l'échelle humaine, on peut remarquer qu'un auteur prolixe comme Balzac, qui a passé sa vie à écrire, à produit une œuvre qui tient en 40 volumes en édition normale.

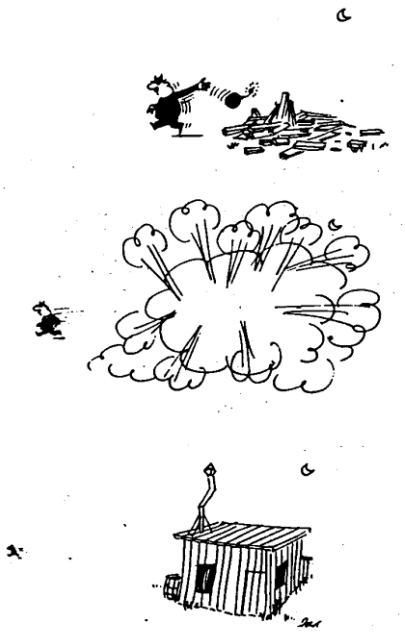
Nous disposons également de statistiques de productivité moyenne des programmeurs depuis plus de 30 ans, dans des milliers d'entreprises. Cette statistique est extrêmement stable, facilement vérifiable pour qui veut bien s'en donner la peine. Elle se situe, pour de grandes masses de programmation, aux alentours de 4.000 LS par programmeur et par an, dans une année normale, avec des congés et des journées normales, et ce quelle que soit la nature du langage de programmation ; soit environ 2 LS à l'heure travaillée.

Un programme de 20.000 LS, i.e. notre livre étalon de 400 pages, c'est le travail de 2-3 bons programmeurs en un an, sachant que la productivité peut varier de 1 à 10, mais que statistiquement on ne trouvera quasiment jamais, dans la réalité des projets, une équipe composée uniquement de très bons programmeurs, à supposer qu'ils arrivent à se supporter. Sur des équipes de 5 à 600 programmeurs, le minimum pour réaliser un

[12] Faire très attention car des chiffres aberrants circulent, comme un Windows à 30 millions de LS !
[13] Voir le site Bull http://www.feb-patrimoine.com/.

système d'exploitation, on n'aura pas plus de 10% de la population étiquetée très bons programmeurs, avec une productivité de 8 à 10.000 LS/an, et un maximum indépassable vers 15-20.000 LS.
On verra un peu plus loin ce qu'il faut entendre par bons ou très bons programmeurs.
Nous savons également par connaissance empirique que l'effectif d'une équipe projet de programmation, à la fois stable et efficace, ne doit jamais dépasser 7-8 personnes pour conserver une bonne communication entre ses membres. Un chef de projet au charisme exceptionnel pourra encadrer 10-15 personnes, mais malheur à celui, moins doué, qui lui succédera ... Les évangélistes des méthodes agiles préconisent 8-10 très bons programmeurs au maximum dans l'équipe.
Une équipe nominale de 7-8 personnes, sur une durée de 1 à 2 ans (au-delà l'équipe se délite et n'arrive pas à maintenir sa motivation) c'est au maximum une production équivalente à 12 personnes/an ; soit environ 50.000 LS ⇒ 2 à 3 livres.
Pour reconstruire un logiciel de 1 million de LS il faudrait donc, au bas mot, une vingtaine d'équipes de programmation. Pour Amadeus, ce serait 2.500 projets à mener de front avec succès, ce qui n'est pas évident.

Nous sommes au cœur de la problématique de l'architecture et de l'urbanisation des systèmes : comment **organiser et gérer de telles masses d'information** pour qu'elles restent compréhensibles et gérables par des humains aux capacités limitées ?



Pour arriver à coordonner le travail de nombreuses équipes, il faut un plan d'ensemble que chaque équipe doit connaître à fond pour préserver la cohérence du projet, et des plans spécifiques à chacune des équipes qui doivent respecter les règles communes. Faute de quoi, c'est le chaos garanti.
L'image ci-contre montre qu'il est très facile de passer d'une maison à un tas de planches, même bien taillées, mais que le processus inverse est parfaitement illusoire. En matière de reconstruction, il n'y a pas de miracle. Chaque pièce doit être taillée conformément à sa fonction et ses contraintes dans la logique d'ensemble. Les ingénieurs constructeurs de ponts nous enseignent que les échafaudages sont aussi importants que la construction elle-même qu'ils rendent possibles.

Reconstruction

Dans une large communauté, le bouche à oreille, la culture du ouï-dire et la palabre, l'à peu près, sont autant de périls mortels qui vont détruire progressivement la capacité de communication et de compréhension de ses membres. Le premier empereur de la Chine impériale, Qin shi Huang di, celui de l'armée des soldats de terre cuite de Xian, en 200 avant JC, avait légiféré en matière d'idéogrammes de façon à codifier strictement l'écriture de ce qui constitue l'un des piliers de la culture chinoise, avec le confucianisme. Peut-être devrions nous regretter comme européens d'avoir abandonné le latin médiéval au 17ème siècle ?!
Dans la communauté du logiciel « libre », on ne plaisante pas avec les standards d'écriture des programmes, sous peine d'excommunication immédiate.
C'est toujours la même idée ⇒ Standardiser pour mieux communiquer.

Le patrimoine applicatif dans le langage du bibliothécaire

Dans le langage du bibliothécaire, tout est livres, rayonnages, étagères, salles, et même bâtiments si la bibliothèque est vaste, etc.
Aujourd'hui, avec le développement des ordinateurs, les systèmes d'information ce sont largement informatisés, et les bibliothèques physiquement visibles sont passées dans la mémoire de nos ordinateurs, mais la « masse » informationnelle est restée la même, car le cerveau humain, lui, n'a pas changé.
Aux programmes dont nous avons parlé ci-dessus, il faut maintenant adjoindre les données que ces programmes manipulent. Les données ont comme caractéristiques principales d'être créées par les opérateurs et/ou les usagers du système, pas par les programmeurs qui ont écrit les programmes. Avec le développement d'Internet dans le grand public, la saisie de l'information se fait désormais directement chez l'usager.
Un grand fichier comme celui des foyers fiscaux, ou des clients de EDF, c'est 20 à 30 millions de références. Si l'on compte une page de texte par référence, environ 4.000 caractères, cela fait, aux normes de l'édition 50 à 75.000 livres.
Le SI de la société Amadeus traite environ 480 millions de requêtes et 3 millions de réservations par jour (ce sont des messages de quelques lignes), soit presque 10 fois plus !
Le SI de la Cnam-TS traite annuellement environ 1,2 milliards de bordereaux. Si on assimile UN bordereau = UNE page, on arrive cette fois à des volumes colossaux équivalents à 3 millions de livres par an. On est cette fois dans l'ordre de grandeur des très grandes bibliothèques.
Tous ces chiffres dépassent totalement les capacités du cerveau le mieux structuré et chacun peut comprendre que le seul viatique est l'organisation, la simplification, la standardisation et la rigueur. Comme dit un[14] des bons auteurs en ingénierie des systèmes complexes : simplifier, simplifier et encore simplifier. Mais comme on l'a déjà dit : comment faire ?

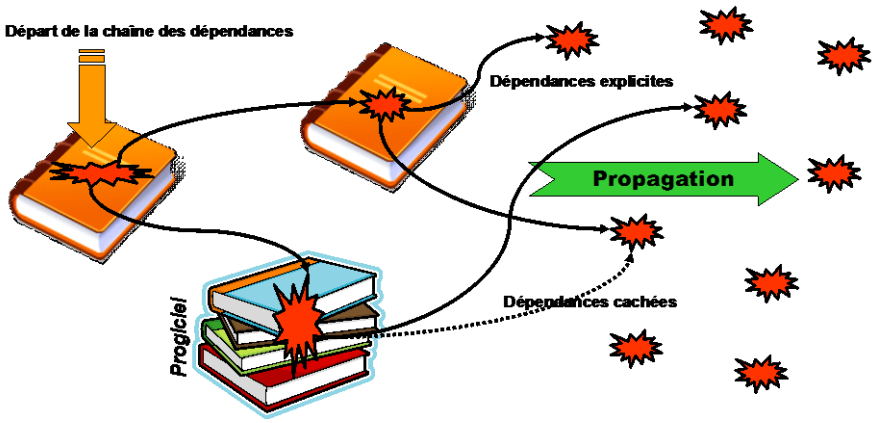
Nous proposons une règle de bon sens : Architecture **plus simple** ⇒ **moins de tests** à développer pour la valider.
Tout ce qui est inutile est nuisible, nous dit la bonne vielle règle du rasoir d'Ockham.

Or nous avons tous une tendance irrépressible à fabriquer de la complexité et du désordre. Il suffit d'être fatigué, dérangé dans son travail ou simplement de se laisser aller. Notre paresse naturelle nous pousse à justifier, souvent avec beaucoup d'énergie, les cas particuliers et le bricolage cognitif, alors que le salut est toujours dans la généricité.
A cette masse d'information se rajoute un autre phénomène qui est le fruit de nos limitations intrinsèques : nous ne sommes pas parfaits, nous faisons tous des erreurs, en plus ou moins grand nombre, mais personne n'échappe à cette loi ergonomique, même les plus grands mathématiciens comme Henri Poincaré, réputé pour en faire beaucoup, mais à sa décharge elles étaient faciles à corriger.

Tout acte humain est soumis à l'erreur et comme on dit : l'erreur est humaine. C'est ce qu'il ne faut jamais oublier quand on veut comprendre quelque chose au logiciel.

[14] E.Rechtin, *System architecting*, Prentice Hall.

Nous avons des statistiques précises sur le nombre d'erreurs commises en moyenne par les opérateurs de saisie : aux alentours de 5 à 10 erreurs par heure d'activité ; beaucoup plus si c'est un usager par définition peu formé, qui fait sa saisie sur Internet. Le fait d'avoir dilué la saisie sur des millions d'internautes a peut-être fait disparaître les coûts apparents de saisie, mais il est certain que le problème de la qualité des données ne pourra qu'empirer, et il est surtout devenu plus diffus.
Cela signifie que les données qui vont être traitées par les programmes contiennent des erreurs, d'où pour les données sensibles des mécanismes de double saisie, basés sur le fait que deux opérateurs qui fonctionnent avec le même taux d'erreur commettront statistiquement le même nombre d'erreurs, mais pas au même endroit. Du point de vue de la qualité des données, c'est une opération payante. La fusion des deux saisies à 10^{-4} fera au final une saisie aux alentours de 10^{-8}.
De toute façon, le programme de traitement devra obligatoirement prévoir la présence d'erreurs qu'il faut considérer comme inévitables, comme le sont les défauts des matériaux du monde physique. Ces erreurs seront distribuées aléatoirement dans le fichier de saisie.
Dans le cas d'interconnections de fichiers, ce qui est aujourd'hui la règle générale, le problème est la propagation de données erronées, avec la circonstance aggravante que plus les machines sont rapides, plus la propagation des erreurs est rapide. Or comme nous l'ont enseigné les logiciens depuis bien longtemps, raisonner juste sur des données fausses peut amener à conclure n'importe quoi. Lorsque l'erreur est constatée, encore faut-il pouvoir remonter à sa source, ce qui exige une traçabilité inverse soignée[15].



Propagation des erreurs et métastases

Si l'architecte ne fait pas le tri entre les données qui s'échangent et celles qui peuvent rester privées le chaos est inéluctable. Là encore la règle est de simplifier, simplifier et encore simplifier, de façon à ne rendre visible à tous que ce qui est indispensable. Le reste doit rester privé non pas par désir de tout cacher mais pour une saine politique de qualité des données.
Enfin, et pour conclure ce point, chacun sait que derrière la structure visible des mots et des phrases d'un livre, se cache toujours une autre structure, invisible, qui sont les relations que tel mot, tel phrase, tel chapitre entretient avec d'autres éléments du livre, voire avec d'autres livres. Ce réseau de relations porte le vrai sens du programme et sa complexité réelle. Dans les livres bien faits, donc plus coûteux, outre la table des matières, le lecteur est aidé par différents index, par des notes en bas de pages, par des

[15] C'est le problème difficile de la réversibilité du calcul ; cf. les travaux de C.H.Bennett, R.Landauer, ...

typographies ad hoc pour naviguer dans ce réseau de relations. En cas de modifications du livre, il est facile de casser certaines de ces relations, et le livre devient progressivement incohérent. C'est bien pire si le livre ne contient aucun index, car alors on modifie à l'aveugle dans l'ignorance du contexte auquel les index, s'ils sont bien conçus, donnent accès.

La psychologie du programmeur et l'impact de la dématérialisation des supports physiques de la programmation

Du coté de la programmation proprement dite, les choses deviennent un peu plus subtiles et il nous faut rentrer dans la psychologie du programmeur. C'est un point fondamental qui a fait l'objet d'études précises dès les années 70, que l'on retrouve dans des approches comme les méthodes agiles, l'« *eXtreme programming* », ou le développement piloté par les tests. En général, les programmeurs bien formés ont une bonne mémoire et raisonnent plutôt bien, mais cela ne les empêchent pas de faire des erreurs ; quant aux autres la plus extrême prudence s'impose ... donc former, former et encore former ; c'est le meilleur investissement.

D'un programmeur qui proclamerait ne pas faire d'erreur, on ne peut tirer que deux conclusions : soit il se vante inconsidérément, soit il s'illusionne, et c'est encore plus grave. Il nous faut donc réfléchir à comment faire le moins d'erreurs possible, et surtout comment les détecter au plus vite. La « pression » par les pairs est un puissant facteur de vigilance, et mécaniquement on commet moins d'erreurs, mais encore faut-il que les pairs soient de bon niveau, et reconnus comme tel. Les communautés du logiciel libre exacerbent ce trait psychologique de reconnaissance par les pairs, comme à l'époque du compagnonnage et de nos cathédrales. Cela se joue dans la durée, et c'est assez efficace.

Là encore nous disposons de statistiques sur plus de trente ans, stables et fiables, et que chaque organisation de développement peut facilement vérifier si elle s'en donne la peine. Généralement, ces statistiques, au départ proposées par la NASA, sont exprimées en nombre d'erreurs par millier de LS (Err/KLS). C'est du simple bon sens, car les erreurs sont dans ce que nous écrivons réellement, lignes de programmes et/ou données saisies, et pas ailleurs. L'acte de traduction est intrinsèquement générateur d'erreurs, et tout programme est une traduction d'un besoin dans quelque chose que l'ordinateur pourra exécuter. C'est l'acte de programmation qui fabrique, par effet de bord, les défauts logiciels consécutivement aux erreurs des programmeurs.

Dans la pratique industrielle, cela donne les résultats moyens suivant :

- Une erreur toutes les 10 lignes source, pour un programme « brut » qui n'a été traité que par les outils classiques des environnements de programmation : compilateurs, éditeurs, etc. ; soit 100 Err/KLS Aux normes de l'édition, c'est cinq erreurs par page.
- Une ou deux erreurs toutes les 1.000 lignes source pour un programme qui a suivi un cycle de validation complet, conforme aux exigences qualité requises par le contrat de service du programme ; soit 1 à 2 Err/KLS. C'est deux erreurs toutes les 20 pages .
- Une erreur toute les 10.000 lignes sources pour des programmes jugés critiques[16] qui ont suivi un cycle de validation poussé (comme par exemple les logiciels de la navette spatiale) ; soit 0,1 Err/KLS. Soit une erreur toute les 200 pages. Pour de tels logiciel, le coût de la validation est de l'ordre de 70% du coût total.

[16] De tels programmes doivent respecter des normes contraignantes comme la DO178 du RTCA/EUROCAE en avionique.

On dispose également de statistiques précises sur les durées moyennes de corrections des erreurs en intégration. Voici celles de Hewlett Packard[17] :

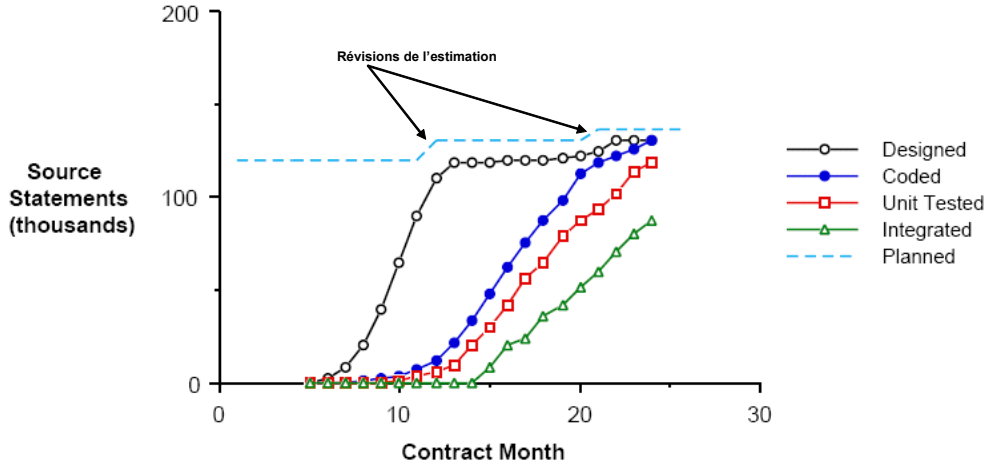
Temps moyen de correction des défauts (soit environ 6h/défaut)
25% des défauts sont diagnostiqués et corrigés en 2h
50% des défauts sont diagnostiqués et corrigés en 5h
20% des défauts sont diagnostiqués et corrigés en 10h
4% des défauts sont diagnostiqués et corrigés en 20h
1% des défauts sont diagnostiqués et corrigés en 50h

Temps moyen de correction des défauts

Pour illustrer concrètement ces chiffres, on peut raisonner avec un programme de 20.000 LS, soit un livre de 400 pages.
Le livre « brut » en sortie des correcteurs orthographiques (lexique et syntaxe) comportera environ 2.000 erreurs dont environ 300 seront découvertes en intégration. En fin de cycle de validation, il ne devra pas contenir plus de 40 erreurs, mais évidemment les programmeurs ne sauront pas où elles se situent dans le programme. Le coût moyens des corrections, en utilisant les statistiques HP sera aux alentours de 1,5 homme×an (30% du coût).
Cela veut dire qu'il aura fallu effectuer plus de 2.000 modifications du programme, car en plus des erreurs, il y aura eu des modifications pour améliorer le style, soit une réécriture quasi complète du programme. Moralité, si les programmeurs ne disposent pas d'un ensemble d'index qui organisent et structurent le texte de leur programme, son évolution dans la durée sera compromise. La construction de ces index est un aspect fondamental de l'architecture et de l'organisation du programme. Pour cela ont été créés des outils comme les dictionnaires de données et les gestionnaires de configuration pour assister les équipes dans ce travail à la fois difficile et fastidieux.
Remarque : Une des plus célèbres erreurs résiduelles est celle qui a occasionné la destruction en vol du premier lanceur Ariane 5. Nichée dans le logiciel de guidage de la centrale de navigation du lanceur, elle était passée inaperçue lors des 40 tirs d'Ariane 4 effectués avec succès. Rien n'avait bougé dans les instructions, seul le comportement du lanceur était différent.

Cela permet de dessiner la courbe de maturité d'un programme que là encore, toute organisation de développement peut vérifier si elle veut s'en donner la peine[18].
La figure ci-dessous, extraite du document de comptage du SEI cité précédemment montre quatre courbes de maturité qui reflètent la progression du travail des programmeurs selon les processus normalisés du cycle de développement logiciel : conception, programmation/codage, tests unitaires, intégration/validation. La forme caractéristique, dite en S, montre la dynamique de la maturité qui n'est jamais linéaire.

[17] R.G.Grady, D.L.Caswell, *Software metrics: establishing a company-wide program* – R.G.Grady, *Practical software metrics for project management and process improvement*, chez Prentice Hall.
[18] Pour des organisations de niveau 3 dans l'échelle CMMI, cette statistique est obligatoire.



Courbes de maturité

Bons et très bons programmeurs

A diverses reprises nous avons parlé de bons et/ou très bon programmeurs. Cette notion recouvre deux facteurs qu'il ne faut pas confondre :

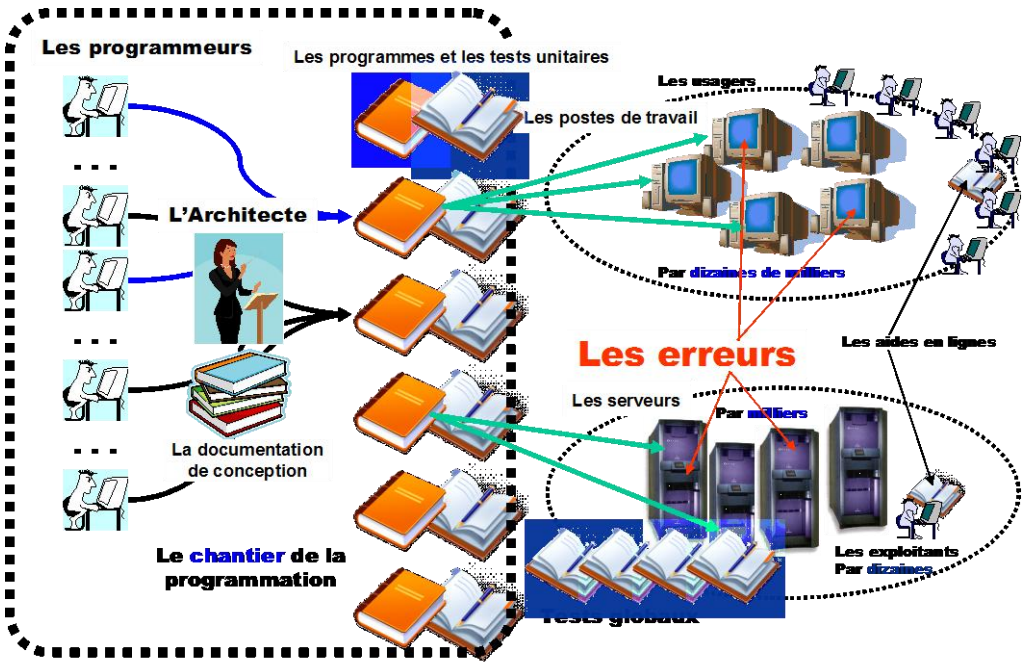
- La compétence du programmeur, qui dépend essentiellement de sa capacité à abstraire et organiser (c'est de l'architecture, au sens litéral) son travail de programmation, en y incluant les tests qui sont une forme de preuve. C'est une caractéristique intrinsèque de sa structure psycho cognitive.
- L'expérience du programmeur, qui ne s'acquiert que sur le terrain, dans la durée, en programmant soi-même, et le plus possible, en expérimentant les bibliothèques d'API des différents langages.

Tout cela prend du temps, et ne s'apprend pas que dans les livres. Une bibliothèque d'API peut compter plusieurs centaines de fonctions et/ou de méthodes. Si on compte un ou deux jours de travail en moyenne par fonction, on comprend tout de suite que le processus de maturité prendra plusieurs années, à supposer que le programmeur ait une excellente mémoire. Aux normes de l'édition, si l'on compte en moyenne 3-5 pages par fonction, il lui faut mémoriser dans le détail, avec exactitude, l'équivalent de 3 à 4 livres[19]. Un très bon programmeur fera de toute façon des erreurs de programmation, mais il saura les corriger rapidement, sans en créer d'autres. Et cela se verra sur les courbes en S.

Le vrai patrimoine applicatif

Les programmes sont la partie émergée et visible du patrimoine applicatif. Pour gérer ce patrimoine dans la durée, il faut des textes qui expliquent d'une part comment c'est fait et comment ça marche, i.e. la conception des programmes ; et d'autre part pourquoi ça marche, i.e. les tests qui nous permettent d'affirmer que le niveau de qualité est correct.
En fait la bibliothèque associée d'un programme contient trois types de livres : la conception, la programmation et les tests. Les outils de gestion ont longtemps été appelé bibliothécaire ou « *librarian* ». Une machine sans le mode d'emploi, sans la documentation de maintenance, a un intérêt limité.
La situation réelle est donc la suivante :

[19] Jusque dans les années 80, un des plus célèbres livre d'algorithmes était le *Art of computer programming*, de D.Knuth, en 3 volumes de 600 pages chacun.



Le vrai patrimoine applicatif

Il est clair que tous ces textes sont interdépendants. Il est facile d'imaginer ce qui va se passer en l'absence de règles partagées par tous les acteurs de l'ingénierie : programmeurs, testeurs, mainteneurs et exploitants ⇒ Un immense chaos.
La taille du texte de conception sera plus petit, dans un rapport de 50 à 100, par rapport au programme, mais plus difficile à élaborer car c'est le résultat du travail de l'architecte.
La taille des tests sera du même ordre de grandeur que le programme pour des programmes simples, mais beaucoup plus gros pour des programmes algorithmiques et/ou en interactions avec leur environnement.
D'où notre critère de simplicité fondé sur les tests : plus simple ⇒ moins de tests.

L'information dans le nano monde, au delà du biologique

Les prouesses de la microélectronique ont rendu invisible la complexité de nos premiers ordinateurs, avec leurs kilomètres de câblages bien visibles dans les armoires et dans les faux planchers des machines des années 70-80. Mais la complexité est toujours là. Un microprocesseur de 2-3 cm^2 comme ceux que nous avons dans nos PC contient dans son cristal de silicium 4 à 5 milliards de transistors et au moins 50 km de câbles microscopiques, inférieur au micron (le millième de millimètre).
Tout le monde a entendu parler des nanotechnologies dont on commence à maîtriser l'ingénierie. On est cette fois aux alentours du millionième de millimètre. A défaut de sculpter les atomes on commence à savoir sculpter les grosses molécules du vivant. Dans chacune de nos cellules, il y a les chromosomes qui sont des molécules d'ADN qui une fois dépliées mesurent près de 2 mètres de long, constituées de 3,4 milliards de protéines formant un message combinant les 4 lettres de l'alphabet du vivant : ATGC dénotant les 4 protéines qui les constituent. A l'échelle du bibliothécaire et de nos livres de 400 pages, cela ferait environ 2.800 livres, dans chacune de nos cellules.
Le cerveau humain est un immense réseau formé d'environ 100 milliards (10^{11}) de neurones, chaque neurone étant relié à 10.000 de ses voisins, soit 1 million de milliards de connexions (10^{15}). Une description du cerveau sous la forme de livres (50 lignes/1

page par neurone[20] et 1 ligne par connexion) nécessiterait 250 millions de livres pour la description fonctionnelle des neurones et 500 millions de livres pour les connexions. On comprend la nécessité des supercalculateurs pour traiter l'information du génome et celle du cerveau.
Cette immense complexité est parfaitement invisible. Pour la percevoir il faut au minimum des microscopes électroniques et des dispositifs permettant de travailler à l'échelle moléculaire, voire atomique. Paradoxalement, plus c'est petit et plus les appareils sont gigantesques, pour culminer avec le LHC du CERN à Genève pour travailler à l'échelle subatomique. Tout cela devrait nous rendre humble et prudent pour ne pas nous laisser submerger par la complexité. D'où l'injonction déjà mentionnée :

Pour espérer maîtriser la complexité, il faut **simplifier**, simplifier et encore simplifier, puis **organiser**, organiser et encore organiser, et enfin **communiquer**, communiquer et encore communiquer, sans oublier la **validation** à tous les niveaux.

Moralité, ce n'est pas parce que c'est petit que c'est simple et facile à comprendre, c'est même tout le contraire car l'observation à l'échelle humaine devient plus difficile. Dans le crash d'Ariane 501[21], il a suffit de quelques lignes de code, dont personne ne s'était souciée, pour provoquer la destruction du lanceur.

Intégrité de la bibliothèque : pas si simple !

La métaphore de la bibliothèque peut nous aider à bien comprendre comment, si l'on n'y prend pas garde, le désordre peut facilement s'instaurer, et plus rapidement qu'on ne l'imagine. Donnons juste un exemple.
L'opération la plus fréquente consiste à extraire un livre d'un rayonnage pour le donner à un emprunteur qui l'a sélectionné grâce à l'index, et son inverse, remettre le livre exactement à sa place après consultation. C'est le `couper/coller` classique.
Une autre opération consiste à transférer tous les livres d'une même catégorie dans un espace plus grand, quand il y a saturation, de façon à les laisser grouper. Ce qui fait qu'un rayonnage qui servait dans un premier temps à des livres sur les systèmes d'information, pourra être réutilisé pour des livres sur les réseaux, voire une catégorie sans rapport avec l'informatique. C'est une `défragmentation`.
Si le personnel n'est pas bien informé de ces changements d'affectation, on peut se retrouver avec un livre sur les systèmes d'information rangé dans une rubrique qui n'a plus rien à voir avec les SI. Si la bibliothèque ne compte qu'un seul bibliothécaire qui fait son travail en séquence, il n'y aura aucun problème, sauf erreur de sa part, mais le service sera lent.
Dans une grande bibliothèque il y aura des dizaines, voire des centaines de bibliothécaires, et beaucoup plus d'interactions, surtout si le directeur de la bibliothèque souhaite ouvrir sa bibliothèque tous les jours, et effectuer les prêts, les restitutions, les réaménagements sans interrompre le service.
Traduit dans le langage informatique, c'est le passage du monde batch centralisé au monde interactif distribué, bien plus complexe.
Les règles de rangement, la localisation physique des espaces de rangement, les index sont des méta données qu'il faut manipuler avec beaucoup de précautions car c'est le lieu de l'intégrité. On comprend tout de suite que si le personnels est mal formé, que si les

[20] Certainement très insuffisant, car chaque cellule est une machine complexe, et a fortiori les neurones.
[21] Voir le témoignage de première main de Michel Turin, dans ARAGO 20, *Application des techniques formelles au logiciel*, OFTA/ Diffusion Lavoisier ISBN 2/906028-06-1.

règles ne sont pas claires, le désordre va s'installer : aucun livre n'a été volé, mais n'étant pas à leur place logique, ils deviennent de ce fait inaccessibles via les index, et invisible pour l'usager. Dans le jargon des architectes de bases de données, l'objet est devenu un fantôme. On comprend également que les modifications de l'index, quand on ajoute de nouveaux livres ou quand on en retire, sont des opérations bien plus risquées qu'un simple prêt. Les bibliothécaires en charge des index doivent être les plus compétents.
Une telle équipe est l'analogue de programmes mal coordonnés, travaillant sur des données mal organisées. Modifier un dictionnaire de données, ou l'annuaire d'adressage d'un réseau nécessite quelques précautions. Toutes ces opérations ont leurs analogues dans la bibliothèque.

Un autre cas intéressant est celui de plusieurs lecteurs L1, L2, etc. empruntant des livres dont ils souhaitent consulter les références en parallèle avec leur lecture. Soit les références A, B, C pour L1, et C, A, D pour L2. Les livres empruntés par L1 et L2 ont donc des références communes. Imaginons la situation où L1 et L2 sont ensembles à la bibliothèque et commencent à lires leurs ouvrages respectifs. On peut donc avoir une séquence temporelle entrelacée, en fonction de la vitesse de lecture et des événements, comme suit :

- L1 emprunte le livre A, premier de sa liste de référence
- L2 emprunte le livre C, premier de sa liste de référence
- L1 emprunte B
- L2 emprunte A qui est déjà emprunté par L1 $\Rightarrow$ L2 s'arrête
- L1 continue sa lecture et veut emprunter C qui est déjà emprunté par L2 $\Rightarrow$ L1 s'arrête

Moralité, les deux lecteurs que les bibliothécaires ont laissé emprunter sans contrainte sont mutuellement bloqués. On peut imaginer une situation chaotique où tout le monde attend tout le monde. Ce genre de conflits, rare en centralisé, devient la règle par défaut dans un système distribué où les ressources sont partagées, ce qui correspond à une situation classique d'interopérabilité. Sans règle, le blocage général est garanti. Ou alors chacun se débrouille et négocie avec l'autre emprunteur, si tant est qu'ils se connaissent, donc le chaos garanti.
Pour y mettre bon ordre, il faut identifier tout ce qui est partageable, et fixer des règles rigoureuses d'emprunts pour terminer tranquillement sa lecture.

L'architecte et le chaos : architecture de la complexité

On aura compris qu'à ce stade, l'architecte est fondamentalement un créateur d'ordre. En se sens, il est urbaniste et architecte. Mais l'ordre qu'il doit créer dans l'ingénierie de l'information est par nature sémantique, fondé sur le « à quoi, à qui ça sert ». La métaphore de la bibliothèque montre que pour offrir un service de qualité, la bibliothèque doit être correctement organisée à partir des contenus, en fonction du service que son directeur souhaite offrir. Elle montre également que le personnel doit être correctement formé et assigné à des tâches en rapport avec son niveau de qualification. Confier la gestion des index à du personnel peu qualifié c'est préparer une catastrophe qui ne manquera pas de se produire.
A contrario, si on est dans le laisser faire et le bon vouloir, sans gouvernance explicite, si l'on fait trop confiance, les livres seront disposés en vrac sur les étagères, au gré des arrivages et des humeurs de chacun. Pour un œil non averti, tout pourra même avoir les apparences d'une bonne gestion.

Les règles de rangement seront dans la tête des bibliothécaires, et non pas dans les procédures, ce qui fait que le départ d'un bibliothécaire effacera ipso facto les connaissances qu'il détenait, augmentant encore le désordre et l'anarchie. Pour rechercher un livre, il n'y aura plus que la connaissance empirique du rangement, avec des scrutations de nombreux ouvrages avant de trouver celui recherché. Tout se ralentira, avec un risque permanent de blocage.
Le patrimoine applicatif, les applications « legacy » comme on dit en franglais informatique, est semblable à une bibliothèque mal organisée, ou en cours de dégénérescence, par non respect des règles d'intégrité qui ont été progressivement oubliées parce que mal documentées et/ou mal expliquées aux nouveaux arrivants.
Mettre de l'ordre dans la complexité, c'est hiérarchiser et organiser les livres en fonction de leurs usages, non pas de leur taille ou de leur couleur. C'est définir et faire appliquer des critères de classement fondés sur les contenus, et non pas sur l'aspect externe. C'est mettre en place des mécanismes qui permettent de s'assurer que les livres sont bien à leur place, quelles que soient les circonstances. C'est aussi standardiser les infrastructures et les plates-formes autant que faire se peut.
Tout le monde peut comprendre que modifier ou emprunter un livre dans un ensemble bien organisé n'aura pas le même coût que dans un fouillis hétéroclite sans structure, et que le risque encouru sera bien plus grand.

La **métaphore de la bibliothèque** et des livres explique de façon simple et pertinente quelques uns des **mécanismes basiques** que l'architecte devra mettre en œuvre pour **maîtriser la complexité**, donc les coûts, dans des **limites qui restent acceptables** humainement et économiquement. En bref, comment être intelligent !

L'ingénieur et le bricoleur

Un clin d'œil ethnologique pour conclure avec un texte de C.Lévi-Strauss, dans *La Pensée sauvage*, Plon.

…
Le bricoleur est apte à exécuter un grand nombre de tâches diversifiées ; mais, à la différence de l'ingénieur, il ne subordonne pas chacune d'elles à l'obtention de matières premières et d'outils, conçus et procurés à la mesure de son projet : son univers instrumental est clos, et la règle de son jeu est de toujours s'arranger avec les « moyens du bord », c'est-à-dire un ensemble à chaque instant fini d'outils et de matériaux, hétéroclites au surplus, parce que la composition de l'ensemble n'est pas en rapport avec le projet du moment, ni d'ailleurs avec aucun projet particulier, mais est le résultat contingent de toutes les occasions qui se sont présentées de renouveler ou d'enrichir le stock, ou de l'entretenir avec les résidus de constructions et de destructions antérieures. L'ensemble des moyens du bricoleur n'est donc pas définissable par un projet (ce qui supposerait d'ailleurs, comme chez l'ingénieur, l'existence d'autant d'ensembles instrumentaux que de genres de projets, au moins en théorie) ; il se définit seulement par son instrumentalité, autrement dit et pour employer le langage même du bricoleur, parce que les éléments sont recueillis ou conservés en vertu du principe que « ça peut toujours servir ».

Par bien des côtés, dans les sciences de l'information, nous sommes encore des « sauvages », et nous en payons le prix. Mais il ne tient qu'à nous de nous civiliser et de mettre un terme au bricolage. Nous en avons tous les moyens, il suffit de se pencher et de ramasser ce que nos prédécesseurs ont accumulé comme savoir et savoir-faire, et de les transmettre correctement, en y intégrant les vraies innovations, rares, par définition.

Bibliographie

Cités dans le texte :
Umberto Eco, *Sémiotique et philosophie du Langage*, PUF – *Kant et l'ornithorynque*, Grasset – *Au nom de la rose*, Grasset.
John von Neumann, *The computer and the brain*, disponible en français ou en anglais.
R.E.Park, *Software size measurement: A framework for counting source statements*, CMU/SEI-92-TR-20 ; site du SEI.
D.Parnas, *On the criteria to be used on decomposing systems in modules*, CACM, 1972.
Jacques Printz, *écosystème des projets informatiques*, Hermès.
Jacques Printz, *Architecture logicielle, concevoir des applications simples, sûres et adaptables*, Dunod.
Jacques Sassoon, *Urbanisation des systèmes d'information*, Hermès, Coll. Management et Informatique, 1998.
John Zachman, *A framework for information systems architecture*, IBM Systems Journal, Vol. 26, N°3, 1987.
Charles H. Bennett, *Logical reversibility of computation*, IBM Journal of research & development, November 1973.

Sur l'architecture et l'urbanisme, on peut lire les ouvrages de Le Corbusier, dont plusieurs sont disponibles en livre de poche. Aux éditions du Moniteur, spécialisées en architecture, on trouve des ouvrages passionnants comme : A.Picon, *L'art de l'ingénieur, constructeur, entrepreneur, inventeur* ; S.Deswarte, B.Lemoine *L'architecture et les ingénieurs, Deux siècles de réalisations* ; G.Ragot, M.Dion, *Le Corbusier en France, Projets et réalisations*.

Sur la psychologie des programmeurs, voir :
W.Humphrey, *Managing technical people*, Addison Wesley.
F.Détienne, *Génie logiciel et psychologie de la programmation*, Hermès.
J.Printz, *Productivité des programmeurs*, Hermès.
G.Weinberg, *The psychology of computer programming*, Silver edition, Dorset House Publishing, 1998.

Sur les statistiques concernant les erreurs, voir :
B.Beizer, *Software testing techniques*, Van Nostrand Reinhold.
P.G. Neumann, *Computer-Related Risks*, Addison-Wesley and ACM Press ; ainsi que le site http://www.csl.sri.com/users/neumann/insiderisks.html
NIST, *The economic impacts of inadequate infrastructure for software testing*, site du NIST, for Dr Gregory Tassey, Senior Economist.

Sur l'ergonomie et les facteurs humains, la sociologie du risque, voir :
R.W.Bailey, *Human preformance engineering*, Prentice Hall.
U.Beck, *Risk society*, SAGE.
R.S.Bridger, *Introduction to ergonomics*, Taylor & Francis.
J.Reason, *L'erreur humaine*, PUF.
F.Vanderhaegen, *Analyse et contrôle de l'erreur humaine*, Hermès.
D.Vaughan, *The Challenger launch decision*, University of Chicago Press.